DISTRIBUTED CLASSIFIER CHAIN OPTIMIZATION FOR REAL-TIME MULTIMEDIA STREAM MINING SYSTEMS

Brian Foo, Mihaela van der Schaar

University of California Los Angeles (UCLA), Dept. of Electrical Engineering (EE), Los Angeles, CA, 90095
Tel: +1-713-818-9047, Fax: +1-310-206-4685, Email: {bkungfoo , mihaela}@ee.ucla.edu

ABSTRACT

We consider the problem of optimally configuring classifier chains for real-time multimedia stream mining systems. Jointly maximizing the performance over several classifiers under minimal end-to-end processing delay is a difficult task due to the distributed nature of analytics (e.g. utilized models or stored data sets), where changing the filtering process at a single classifier can have an unpredictable effect on both the feature values of data arriving at classifiers further downstream, as well as the end-to-end processing delay. While the utility function can not be accurately modeled, in this paper we propose a randomized distributed algorithm that guarantees almost sure convergence to the optimal solution. We also provide results using speech data showing that the algorithm can perform well under highly dynamic environments.

Keywords: Distributed systems, Queuing theory, Multimedia stream mining, Multiagent systems

1. INTRODUCTION

Recently, data mining applications have emerged that require operations such as classification, filtering, aggregation, and correlation over high-volume, continuous multimedia streams in real time [1]. Due to the naturally distributed set of data sources and jobs, as well as high computational burdens for the analytics, distributed mining systems have been recently developed [2]. These systems leverage computational resources from a set of distributed processing nodes and provide the framework to deploy and run stream mining applications on various resource topologies [2]. A key challenge in distributed, real-time multimedia stream mining systems is how to cope effectively with system overload, while maintaining high performance under delay constraints. A commonly used approach is load-shedding, where algorithms determine when, where, what, and how much data to discard given the observed data characteristics, e.g. burst, and desired Quality of Service (QoS) requirements. Recent work on intelligent load shedding [3] attempts instead to maximize certain Quality of Decision (QoD) measures based on the predicted distribution of feature values in future time units. However, such load shedding algorithms are limited by the fact that the algorithms consider only local information and metrics pertaining to single classifiers. Because distributed systems process data using entire sequences (or chains) of classifiers [2], the performance of load shedding can be highly sub-optimal with regards to the end-to-end performance (and delay) across several classifiers, as data discarded at one stage may be essential for a downstream classifier.

Optimally configuring classifiers across an entire chain is a difficult problem, not only due to the computational complexity, or the required coordination between autonomous sites, but also due to informational constraints. In most distributed stream processing systems, the analytics are distributed across autonomous sites that are neither willing to share their analytics, nor willing to provide a repository to store an entire collection of data across multiple sites [4].

To overcome the unpredictable nature of the system, in this paper, we propose a low-complexity, stochastic algorithm for optimizing the performance of a real-time, distributed binary filtering classifier system. While the proposed algorithm requires periodic information exchange across the network, it requires no coordination between individual classifiers. Moreover, using a multimedia speaker verification application, we show experimentally that the algorithm can quickly adapt to new environments.

The paper is organized as follows: Section II introduces the model used for classifiers and classifier chains, and derives a utility function for real-time multimedia stream mining applications. Section III discusses the limitations that prevent accurate modeling of the utility function, and proposes the stochastic algorithm for configuring a chain of classifiers. Section IV provides results based on a speech data mining application, and Section V concludes the paper.

2. DISTRIBUTED STREAM PROCESSING SYSTEM MODEL

2.1 Characterizing Binary Classifiers and Classifier Chains

A binary classifier v_i processes an input stream by classifying each stream data object (SDO) X_i as belonging to a positive class H_i, or a negative class $\overline{\mathrm{H}}_i$. If the SDO is identified as positive, it is outputted as $\hat{X}_i$ and forwarded to its next destination. Otherwise, the SDO is dropped from the stream. Given X_i and $\hat{X}_i$, the proportion of correctly forwarded samples is captured by the *probability of detection* $P_i^D = \Pr\left\{\hat{X}_i \in \mathrm{H}_i \mid X_i \in \mathrm{H}_i\right\}$, and the proportion of incorrectly forwarded samples is captured by the *probability of false alarm* $P_i^F = \Pr\left\{\hat{X}_i \in \mathrm{H}_i \mid X_i \notin \mathrm{H}_i\right\}$.

Suppose that the input stream to classifier v_i has *a priori probability* (APP) π_i of being positive. The probability of forwarding an SDO to the next classifier can be given by:

$$\ell_i = \pi_i P_i^D + (1-\pi_i) P_i^F . \tag{1}$$

Moreover, the probability of *correctly forwarding* data to the next classifier is:

$$\wp_i = \pi_i P_i^D . \tag{2}$$

A filtering classifier is often designed such that the probability of detection is maximized subject to a false alarm probability constraint [6]. Hence, by varying the false alarm constraint, different probabilities of detection can be obtained, and different volumes of the stream can be forwarded. Additionally, we assume that each classifier operates at a fixed complexity level, such that the $\left(P_i^F, P_i^D\right)$ curve is fixed (See [7] for an example of different curves at different complexities.). Hence, given the APP π_i, ℓ_i and $\wp_i$ are deterministic functions of P_i^F.

A binary classifier chain is shown in Fig. 1, where an SDO forwarded from a classifier node v_i is inputted to the next classifier node v_{i+1} in the chain. SDOs generated by the source v_0 are only received at the terminal v_N if they are forwarded by every classifier in the chain (i.e. they are relevant for the application).

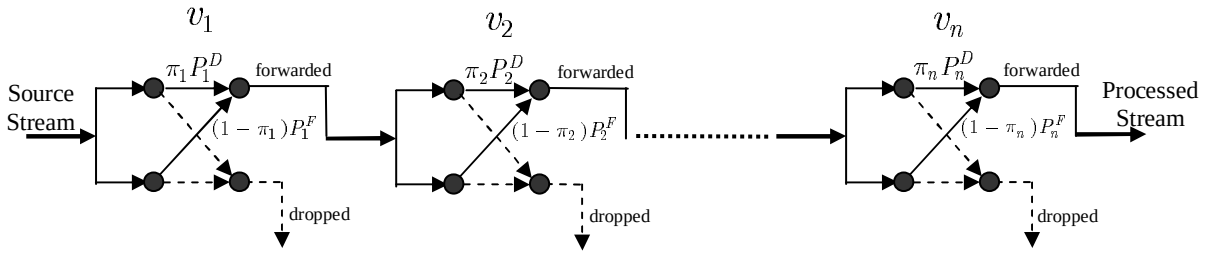


Fig. 1. Classifier chain with probabilities labeled on each edge.

2.2 Data Object Traffic Model and Delay Analysis

Because the end-to-end delay for each received SDO is important for real-time applications [13], we provide in this section an analytical queuing model for a classifier chain to estimate the delay distribution. Since a distributed stream processing system is typically networked, we follow the $M/M/1$ queuing model often used for networks and distributed stream processing systems [11]. Hence the resulting output to each next-hop neighbor can also be modeled by a Poisson process [10]. The delay associated with a single classifier v_i then follows an exponential distribution [8]:

$$D_i \sim p_{D_i}(t) = (\mu_i - \lambda_i) e^{-(\mu_i - \lambda_i)t}, \ \lambda_i \le \mu_i . \tag{3}$$

Because the output of an $M/M/1$ system has i.i.d. interarrival times [9], the delays for each classifier in a classifier system, given the arrival and service rates, are independent. Thus, the total end-to-end delay for the classifier chain in Fig. 1 is:

$$D = \sum_{i=1}^{n} D_i \,, \tag{4}$$

and follows a gamma distribution with a moment generating function given by:

$$\Phi_D(r) = E\left[e^{rD}\right] = E\left[\exp\left(r\sum_{i=1}^{n} D_i\right)\right] = \prod_{i=1}^{n}\left(\frac{\mu_i - \lambda_i}{\mu_i - \lambda_i - r}\right). \tag{5}$$

2.3 Objective Function for a Filtering Classifier Chain

In most prior works, the performance of a stream mining application is evaluated based on a linear cost in the fraction of misses and false alarms [12]. Based on a weight ratio θ_i between the cost of misses and cost of false alarms, a performance metric for classifier v_i given by:

$$\begin{aligned} F_i &= \wp_i - \theta_i(\ell_i - \wp_i) \\ &= \pi_i P_i^D - \theta_i(1-\pi_i)P_i^F \end{aligned} \tag{6}$$

Based on the chain of n classifiers $v_1, v_2, \ldots, v_n$, shown in Fig. 1, where each classifier v_i has a weighting parameter θ_i depending on the function being performed, we define the total performance to be the product of all performance reductions induced by each classifier node in the chain, i.e.:

$$F = \prod_{i=1}^{n} F_i = \prod_{i=1}^{n} (\wp_i - \theta_i(\ell_i - \wp_i)) \,. \tag{7}$$

Note that we must set each $F_i = [F_i]^+$ (if the performance metric is negative, we set it to 0, such that no benefit is derived) since negative performance reductions could otherwise be multiplied to produce positive performance.

Because the usefulness of data for real-time applications decays as a function of the processing time, we introduce a delay penalty factor $G(D) = e^{-\varphi D}$ for a processing delay of D. The decaying exponential form is versatile, since the parameter φ can be adjusted to capture applications with a wide range of delay sensitivities (i.e. when φ is large, the application has high delay-sensitivity, while when φ is low, the application is not very delay-sensitive). Based on our traffic model in the previous subsection, the delay follows a gamma distribution. The average delay penalty is then the moment generating function for D, evaluated at $-\varphi$ (See (5)). Multiplying together all the penalties, the stream utility maximization problem can be given by:

$$\begin{aligned} \max_{\mathbf{P}^F \forall v_i \in V} Q(\mathbf{P}^F) &= \max_{\mathbf{P}^F} F \cdot \Phi_D(-\varphi) = \max_{\mathbf{P}^F} \prod_{i=1}^{n} [\wp_i - \theta_i(\ell_i - \wp_i)]^+ \prod_{i=1}^{n} \left[\frac{\mu_i - \lambda_i}{\mu_i - \lambda_i + \varphi}\right]^+ . \\ \text{s.t.} \quad & \mathbf{0} \le \mathbf{P}^F \le \mathbf{1} \end{aligned} \tag{8}$$

3. DISTRIBUTED ALGORITHM FOR CLASSIFIER CHAIN CONFIGURATION

3.1 Discussion of Limitations

Before proposing any solutions, a fundamental question must be asked: *Can all the necessary information be gathered to solve the utility maximization problem in* (8)*?* In particular, the utility maximizing objective function relies on obtaining the terms $\wp_i$, ℓ_i, and λ_i, which are all functions of the APP π_i, the false alarm configuration P_i^F, and the detection curve P_i^D across different classifiers. While classifiers may be willing to provide information about P_i^F and P_i^D, the APP π_i at every classifier v_i is, in general, a complicated function of the false alarm probabilities of all previous classifiers, i.e. $\pi_i = \pi_i\left(\mathbf{P}_x^F\right)_{x \in \mathrm{anc}(i)}$. This is because setting different thresholds for the false alarm probabilities at previous classifiers will affect the incoming source distribution to classifier v_i. As discussed in the introduction section, the analytics are not shared between classifiers, and constructing a joint classification model by inference requires

intolerable time and complexity, especially for dynamic systems. Hence, for practical scenarios, the objective function $Q(\mathbf{P}^F, \mathbf{B})$ is unknown.

3.2 Decomposing the Utility Function into a Distributed Form

While the utility can not be explicitly formulated as a function of $\mathbf{P}^F$ due to informational constraints, the utility at any given time instant can still be estimated by exchanging information based on information available to each classifier (See. First, the average service rate μ_i is fixed (*static*) for each classifier and can be exchanged with other classifiers upon system initialization. Second, the arrival rate into classifier v_i, λ_i, can be obtained by simply measuring (or *observing*) the number of SDOs in the input stream. Finally, the goodput and throughput ratios $\wp_i$ and ℓ_i are functions of both the configuration P_i^F and the APP π_i. The configuration is set by the classifier and therefore known, while the APP can be estimated from the input stream. Based on the locally available information, we can decompose the overall utility function into a product of *locally observable* utility functions:

$$\begin{aligned} Q(\mathbf{P}^F) &= \prod_{i=1}^{n} (\wp_i - \theta_i(\ell_i - \wp_i)) \prod_{i=1}^{n} \left(\frac{\mu_i - \lambda_i}{\mu_i - \lambda_i + \varphi} \right) \\ &\approx \underbrace{\left(\frac{\mu_1 - \lambda_1}{\mu_1 - \lambda_1 + \varphi} \right)}_{\text{constant}} \prod_{i=1}^{n-1} \underbrace{\left(\tilde{\wp}_i - \theta_i\left(\tilde{\ell}_i - \tilde{\wp}_i \right) \right) \left(\frac{\mu_{i+1} - \tilde{\lambda}_i \tilde{\ell}_i}{\mu_{i+1} - \tilde{\lambda}_i \tilde{\ell}_i + \varphi} \right)}_{\text{known at } v_i} \underbrace{\left(\tilde{\wp}_n - \theta_n\left(\tilde{\ell}_n - \tilde{\wp}_n \right) \right)}_{\text{known at } v_n}, \\ &= K \prod_{i=1}^{n} \tilde{Q}_i\left(P_i^F \right) \end{aligned} \tag{9}$$

where the local utility functions can be determined by each classifier:

$$\begin{aligned} \tilde{Q}_i\left(P_i^F \right) &= \left(\tilde{\wp}_i - \theta_i\left(\tilde{\ell}_i - \tilde{\wp}_i \right) \right) \frac{\mu_{i+1} - \tilde{\lambda}_i \tilde{\ell}_i}{\mu_{i+1} - \tilde{\lambda}_i \tilde{\ell}_i + \varphi}, \ 1 \le i < n, \\ \tilde{Q}_n\left(P_n^F \right) &= \left(\tilde{\wp}_n - \theta_n\left(\tilde{\ell}_n - \tilde{\wp}_n \right) \right). \end{aligned} \tag{10}$$

Hence, rather than exchanging information about every single observed parameter, classifier v_i needs only exchange its local utility at a given time t, $Q_i\left(P_i^F(t) \right)$, to construct the global utility at time t, $Q\left(\mathbf{P}^F(t) \right) = \prod_{i=1}^{n} Q_i\left(P_i^F(t) \right)$. The information observed and exchanged are highlighted in Fig. 2.

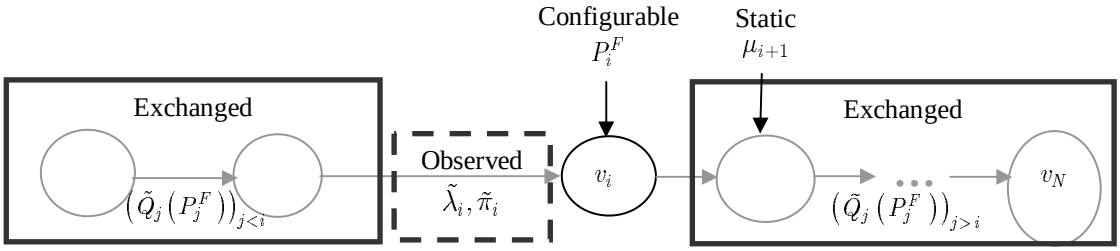


Fig. 2. The various parameters in relation to v_i.

3.3 Safe Experimentation Algorithm for Finite Sets of Configurations

As an alternative to the model-based approaches, we introduce a low-complexity, *model-free* learning approach called *Safe Experimentation* [14] for choosing the best configuration out of a finite set of configurations. Safe experimentation was first proposed for large, distributed, multi-agent systems, where each player is unable to observe the actions of all other players and hence can not build a model of other players. The player therefore adheres to a "trusted" action, but occasionally "explores" a different action in search of a potentially better action. The safe experimentation algorithm for the stream processing system is given as follows:

1) Initialization: At time $t = 0$, each classifier randomly selects a configuration $P_i^F(0)$ from a discrete set of configurations A_i, which is set as the baseline configuration $P_i^{F,b}(1)$. After exchanging information about the derived

local utilities from the initial configurations, each classifier's baseline utility at time 1 is initialized as $u^b(1) = Q\left(\mathbf{P}^F(0)\right)$.

2) Configuration Selection: At each subsequent time step, each classifier selects his baseline configuration with probability $(1-\varepsilon_t)$ or experiments with a new random configuration with probability ε_t. Hence, $P_i^F(t) = P_i^{F,b}(t)$ with probability $(1-\varepsilon_t)$, and $P_i^F(t)$ is chosen uniformly over A_i with probability ε_t. ε_t is denoted the *exploration rate* at time t.

3) Baseline Configuration and Baseline Utility Update: Each classifier obtains the utility at time t, $Q\left(\mathbf{P}^F(t)\right)$, by multiplying the exchanged local utilities from each classifier $Q_i\left(\mathbf{P}^F(t)\right)$. The utility is then compared with the baseline utility $u^b(t)$, and the baseline configuration and utility are updated as follows:

$$P_i^{F,b}(t+1) = \begin{cases} P_i^F(t), & Q\left(\mathbf{P}^F(t)\right) > u_i^b(t) \\ P_i^{F,b}(t), & Q\left(\mathbf{P}^F(t)\right) \le u_i^b(t) \end{cases}, \tag{11}$$

$$u_i^b(t+1) = \max\left(u_i^b(t), Q\left(\mathbf{P}^F(t)\right)\right), \tag{12}$$

4) Return to step 2 and repeat.

The reason why this learning algorithm is called "Safe Experimentation" is because the baseline utility is non-decreasing with respect to time, and hence the performance of the algorithm only improves over time. Note that our stream processing system falls under the category of a common interest game [15], where distributed classifiers (i.e. players) want to configure themselves (i.e. perform actions) to maximize the same unknown utility function. A strong result about convergence can be proven for this scenario, as shown below.

Theorem 1: The following two conditions for the exploration rate ε_t are sufficient to guarantee that the Safe Experimentation algorithm for common interest games converges to the global optimal solution with probability 1:

$$\lim_{t\to\infty} \varepsilon_t = 0, \tag{13}$$

$$\lim_{t\to\infty} \prod_{\tau=1}^{t} \left[1 - \left(\frac{\varepsilon_\tau}{|A_1|}\right)\left(\frac{\varepsilon_\tau}{|A_2|}\right)\cdots\left(\frac{\varepsilon_\tau}{|A_n|}\right)\right] = 0. \tag{14}$$

Proof: The proof is similar to the proof for Theorem 3.1 in [14]. First, the exploration rate must converge to 0 as $t \to \infty$, as indicated by (13), such that the algorithm will play its baseline configuration with probability 1. Moreover, note that each multiplicative term in (14) represents the probability that the joint configuration played at time τ is *not* the optimal joint configuration, *or* all classifiers experimented at time τ. Hence, the left-hand side of (14) is an upper bound on the probability that the optimal joint configuration is *not* played before time t. Thus Eq. (14) provides a sufficient condition for the optimal joint configuration to be eventually played with probability 1. ■

3.4 Continuous Relaxation and Reducing Convergence Time

Note that the safe experimentation algorithm is guaranteed to find a best (discrete) configuration with probability 1, given a proper exploration rate over time, such as $1/\sqrt[n]{t}$. However, the convergence time for the Safe Experimentation algorithm can be (loosely) bounded below by $|\mathcal{A}| = |A_1||A_2|...|A_n|$, which is the expected time for finding the optimal solution via i.i.d. uniform sampling (i.e. $\varepsilon_t = 1, \forall t$). Hence, Safe Experimentation is limited by its long convergence time when configurations are finely quantized.

We propose a solution for continuous configuration sets, which also overcomes the slowness of the convergence time for the discrete Safe Experimentation algorithm. In this algorithm, we combine a uniform random search for a baseline configuration with a *randomized* local search algorithm around the baseline configuration:

1) Initialization: At time $t = 0$, each classifier randomly selects a configuration $P_i^F(0)$ from a feasible (or operational) subset of the interval $(0,1)$, which is then set as the baseline configuration $P_i^{F,b}(1)$. After exchanging

information about the derived local utilities from the initial configurations, each classifier's baseline utility at time 1 is initialized as $u^b(1) = Q\left(\mathbf{P}^F(0)\right)$.

2) Configuration Selection: At each subsequent time step, each player selects his baseline configuration with probability $(1-\varepsilon_t)$ or experiments with a new random configuration with probability ε_t. **If the baseline configuration is selected, it is *perturbed* by a small random variable (e.g. Gaussian, uniform, etc.)** $Z_i(t)$**.** Hence $P_i^F(t)$ is chosen uniformly over the feasible configuration space with probability ε_t, and $P_i^F(t) = P_i^{F,b}(t) + Z_i(t)$ with probability $(1-\varepsilon_t)$, where $Z_i(t)$ is a zero-mean random variable, with $\lim_{t\to\infty} Z_i(t) = 0$.

3) Baseline Configuration and Baseline Utility Update: Each player compares the utility received, $Q\left(\mathbf{P}^F(t)\right)$, with his baseline utility $u^b(t)$, and updates his baseline configuration and utility as follows:

$$P_i^{F,b}(t+1) = \begin{cases} P_i^F(t), & Q\left(\mathbf{P}^F(t)\right) > u_i^b(t) \\ P_i^{F,b}(t), & Q\left(\mathbf{P}^F(t)\right) \le u_i^b(t) \end{cases}, \tag{15}$$

$$u_i^b(t+1) = \max\left(u_i^b(t), Q\left(\mathbf{P}^F(t)\right)\right). \tag{16}$$

4) Return to step 2 and repeat.

If the size of the local search random perturbations do not decay too quickly (e.g. $E\left[|Z_i(t)|^2\right] = K/t^2$, where $K > 0$ is a constant), it can be shown that (in a stationary setting) the local search algorithm eventually converges to a local maximum with probability 1 [16]. While the exploration rate can be set sufficiently high to ensure eventual convergence to the global maximum with probability 1, it is impossible to guarantee both *fast and sure* convergence in a dynamic environment. Nevertheless, Lipschitz-continuous utility functions often yield similar utilities for configurations within small neighborhoods of each other. Because of these correlations between the utility of nearby points in the joint configuration set, some methodologies can be considered for dynamic environments. For example, it is useful to have a very high exploration rate at the beginning of the algorithm, such that a good baseline configuration can be found as a starting point for local search. During later iterations, a very low exploration rate is preferable, such that the local search algorithm can perform "refined" exploration around a good baseline point until stream characteristics change again.

4. SIMULATIONS

The test data consisted of cepstrum tuples from speech signals taken from utterances of the Japanese vowels 'a' and 'e' from 8 different speakers (See [5] for more details.). By sampling different portions of the cepstrum data, we were able to cluster each speakers hierarchically, such that a sequence of classifiers can be used to sequentially filter out speech signals from irrelevant speakers. For example, if the speaker of interest is speaker 1, the data may be first split into two speaker groups {1,2,7,8} and {3,4,5,6} using classifier v_1, then split from {1,2,7,8} into {1,2} and {7,8} using v_2, and finally {1,2} into {1} and {2} using v_3. We obtained a Gaussian mixture model (GMM) based on prior training data (270 speech samples), performed optimization based on a test data stream that comprises of a total of 370 speech data objects divided among the 8 speakers. We also constructed a synthetic speech data set that provided a uniformly distribution along the feature values. The results are shown in Table 1 for synthetic data (assuming the speech samples are distributed across the entire spectrum uniformly) and Table 2 for real speech test data. In our experiments, the randomized local search was performed using a Gaussian random variable with variance that decayed on the order of $O\left(1/t^2\right)$, where t is the iteration number. Note that we provided an approximate global maximum, which was obtained by sampling over 64000 joint configurations of the 3 classifiers. The approaches are compared against a data-independent load shedding scheme, and an "intelligent load shedding" scheme that optimizes the local utility function.

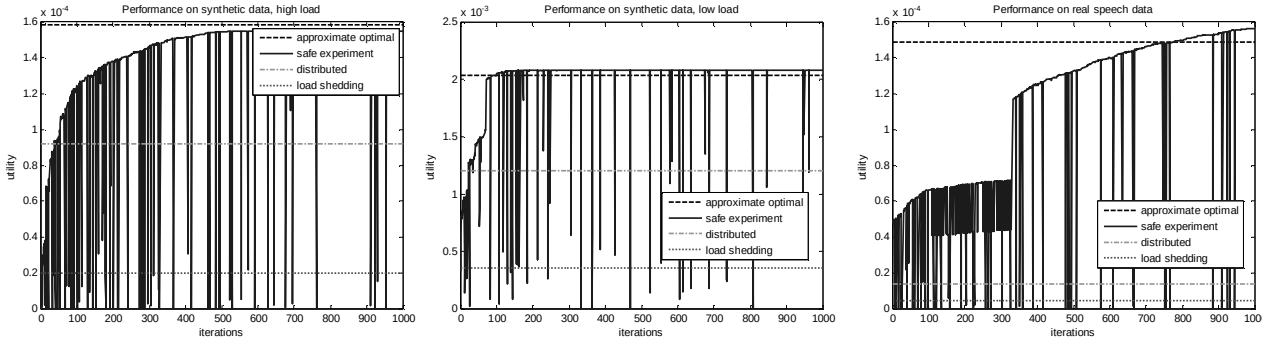
As can be seen from the results for both synthetic data and real speech data (Fig. 3), the continuous Safe Experimentation with random local search greatly outperforms the local utility-based load shedding algorithm, which outperforms the data-independent load shedding scheme. In fact, it can be seen that our algorithm also uses far fewer than 64000 iterations to find the solution that is *better* than the best sampled discrete configuration point!

Table 1 Performance comparison for synthetic data, normalized to the global maximum.

	shedding	distributed	safe exp. (1000 iterations)	approx. global max
low load, low delay-sensitivity	0.1259	0.5806	0.9789	1.0000
high load, high delay-sensitivity	0.2701	0.5776	0.9999	0.9816

Table 2 Performance comparison for speech data test set.

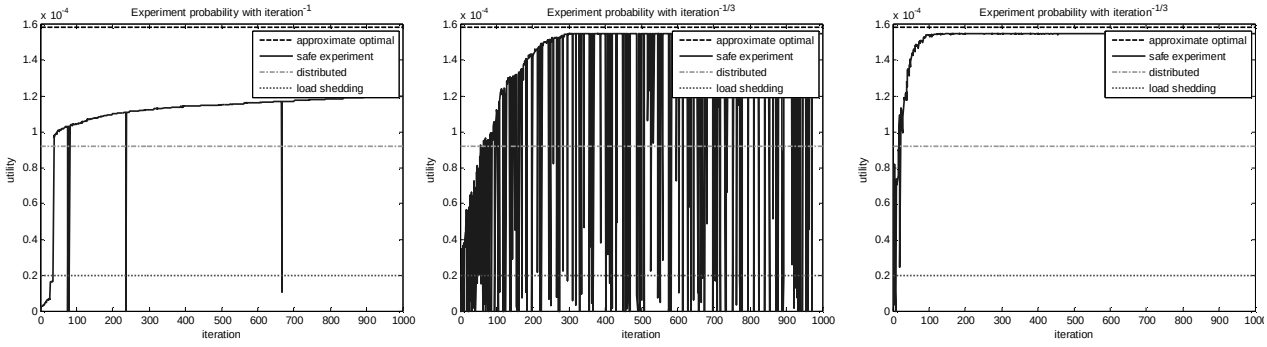
	shedding	distributed	safe exp. (1000 iterations)	approx. global max
low load, low delay-sensitivity	0.0900	0.0814	1.0000	0.9715
high load, high delay-sensitivity	0.0283	0.0893	1.0000	0.9556



(a) (b) (c)

Fig. 3. Comparison of utility versus iteration for load-shedding, distributed, and safe experimentation algorithms under: (a) synthetic data, low load and low delay-sensitivity, (b) synthetic data, high load and high delay-sensitivity, (c) a real speech data stream. The exploration rates in these experiments were set to $1/\sqrt{t}$.

Secondly, we analyzed the adaptation time for Safe Experimentation using synthetic data. In particular, we considered the cases where each classifier experiments with probability $1/t$, $1/\sqrt[3]{t}$, and $t^{-t/r}$, where r indicates the approximate number of explorations performed at the beginning. Some sample curves are shown in Fig. 4 to highlight the rate of adaptation. In particular, note that the exploration rate is very low with $1/t$ and is insufficient for finding the global optimal utility. On the other hand, using $1/\sqrt[3]{t}$, which satisfies the minimal exploration rate condition in Theorem 1, is sufficient for finding the global optimal point, but the exploration rate decays very slowly, and even up to the 1000th iteration. Finally, for $t^{-t/50}$ (as shown in Fig. 4c), frequent exploration is performed during the first few iterations, while local search dominates the later iterations. Fig. 5 shows that for the exploration rate $t^{-t/50}$, in most cases the algorithm adapts quickly to dynamic environments, where we have set the stream characteristics (i.e. fraction of speech signals from each user) to change between every 50 and 100 iterations. Overall, our experiments verify our intuition that, during the first few iterations, it is important to explore frequently to find a good baseline configuration, while for later iterations, "playing it safe" by using randomized local search performs better.



(a) (b) (c)

Fig. 4. Comparison of utility versus iteration for load-shedding, distributed, and safe experimentation algorithms under: (a) Comparison of adaptation times for exploration rates (a) $1/t$, (b) $1/\sqrt[3]{t}$, and (c) $t^{-t/50}$.

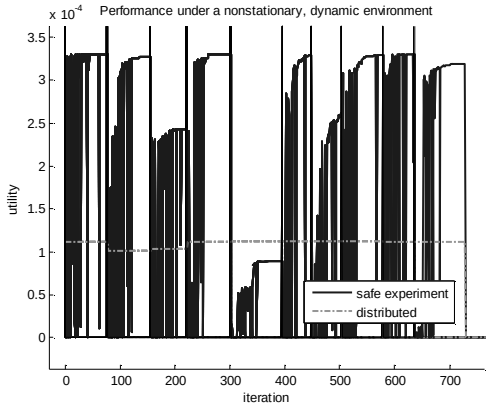


Fig. 5. Dynamic adaptation results of Safe-Experimentation with exploration rate $t^{-t/50}$ for non-stationary streams. Vertical lines indicate the arrival of new stream characteristics.

5. CONCLUSIONS

In this paper, we showed that by exchanging some local performance metrics between classifiers, each classifier can run a low complexity, randomized distributed algorithm that converges "almost surely" to an optimal or near optimal system configuration, even when the global utility function can not be analytically determined. We also provided some insights into the tradeoffs between information availability, complexity, convergence rate, and dynamics in a classifier system, and confirmed these insights through simulations. Most importantly, our solution framework can be extended and applied to any informationally-distributed network or parallel computing system where individually configurable entities have the common goal of optimizing system performance.

REFERENCES

[1] M. Shah, J. Hellerstein, M. Franklin, "Flux: An adaptive partitioning operator for continuous query systems," *International Conference on Data Engineering (ICDE)*, 2003.

[2] L. Amini, H. Andrade, F. Eskesen, R. King, Y. Park, P. Selo, C. Venkatramani, "The stream processing core," *Technical Report RSC 23798*, Nov. 2005.

[3] Y. Chi, P. Yu, H. Wang, R. Muntz, "Loadstar: A load shedding scheme for classifying data streams," *International Conference on Data Mining*, Oct. 2005.

[4] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. Zdonik. "Scalable Distributed Stream Processing," Proc. of CIDR, Asilomar, CA, Jan. 2003.

[5] M. Kudo, J. Toyama and M. Shimbo, "Multidimensional Curve Classification Using Passing-Through Regions," *Pattern Recognition Letters*, Vol. 20, No. 11-13, pp. 1103-1111, 1999. (See: http://kdd.ics.uci.edu/databases/JapaneseVowels/JapaneseVowels.html for the data set.)

[6] A. Hero, J. Kim, "Simultaneous signal detection and classification under a false alarm constraint," *ICASSP*, 1990.

[7] A. Pentland, B. Moghaddam, T. Starner, "View-Based and Modular Eigenspaces for Face Recognition," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1994.

[8] D. Gross and C. Harris, *Fundamentals of Queueing Theory*, New York: Wiley-Interscience, 1997.

[9] P. Burke, "The Output of a Queuing System," *Operations Research* 4, 699, 1956.

[10] R.G. Gallager, *Discrete Stochastic Processes*, Kluwer, Dordrecht, 1996.

[11] S. Viglas, J. Naughton, "Rate-based query optimization for streaming information sources," *SIGMOD*, 2002.

[12] M. Ciraco, M. Rogalewski, G. Weiss, "Improving classifier utility by altering the misclassification cost ratio," *1st International Workshop on Utility-based Data Mining*, 2005.

[13] D. Abadi, D. carney, U. Centintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, "Aurora: a new model and architecture for data stream management," *VLDB Journal*, 2003.

[14] J. Marden, H. Young, G. Arslan, J. Shamma, "Payoff Based Dynamics for Multi-Player Weakly Acyclic Games," submitted to *SIAM Journal on Control and Optimization, special issue on Control and Optimization in Cooperative Networks*, 2007.

[15] D. Fudenberg, J. Tirole, *Game Theory*, The MIT Press, 1991.

[16] R. Horst and P.M. Pardalos, *Handbook of Global Optimization*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.